CSCI 3104 Spring 2014 Problem Set 2
Andrew Howe (08/25)
Ellie Daw (05/31)

1. *Professor Snape has n computer chips that are supposedly both identical and capable of testing each other's correctness. Snape's test apparatus can hold two chips at a time. When it is loaded, each chip tests the other and reports whether it is good or bad. A good chip always reports accurately whether the other chip is good or bad, but the answer of a bad chip cannot be trusted.*

   (a) *Prove that if n=2 or more chips are bad, Snape cannot necessarily determine which chips are good using any strategy based on this kind of pairwise test.*

   Given that the chips are conspiring against the Professor we know that to fool the Proffesor they would declare a bad chip as good and a good chip as bad. Because the chips act in the opposite way of the good chips at all times it is impossible to tell which chips are the good chips and which chips are the bad chips.

   (b) *Consider the problem of finding a single good chip from among the n chips, assuming that more than $n/2$ of the chips are good. Prove that $n/2$ pairwise tests are sufficient to reduce the problem to one of nearly half the size.*

   When Proffesor Snape tests a chip he discards any chip that does not result in both chips reporting that the other is good. Now when their are an odd amount of chips we know that the relation of good chip pairs to bad chip pairs must follow $2p + 1 >= p$ which means we can represent the total number of chips as $4n + 1$ we set aside the chip with no pair and test all the other chips discarding all that do not both report good. We are now left with at most $4n$ chips in our set if the chip that we set aside is good we know we must have atleast 2n good chips, if the chip we set aside is bad then in our remaining set we must have at least $2p + 1$ good chips then we take one chip from each good pair reducing the set to n/2 continuing this process essures that our set has been cut in half but still contains a majority of good chips. Similarly if we even amount of chips if we continue this process we are left with a set that must contain two good chips.

(c) *Prove that the good chips can be identifed with $O(n)$ pairwise tests, assuming that more than $n/2$ of the chips are good. Give and solve the recurrence that describes the number of tests.*

The recurrence that describes the number of tests is given by $T(n) = T(n/2)+n/2$ because each time we are performing $n/2$ tests and then are left with a set that is of at most half the size and requires at most $n/4$ operations until we are left with only one good chip so the complete recurrence is

$$T(n) = T(n/2) + n/2$$
$$T(1) = 0$$

We can solve this recurrence through substitution where we are guessing that $T(n) = O(n)$ subtracting a lower order term d we can show that

$$T(n) = T(n/2) + n/2$$
$$T(n) = c[n/2] + n/2$$

which does imply that $T(n) <= cn$ for any appropriate choice of c.

2. *Problem 2*

(a)

At the worst case the operations contained in the inner loop are n operations. As the loop runs $n-1$ times we now have $(n-1)n$. This loop is a nested loop that runs n times leaving us with $n((n-1)n) = n^3 - n^2$ The bound on the number of operations performed by the algroithm is $O(n^3)$.

(b)

We can show this by taking the limit of $\frac{n^3-n^2}{n^3}$ which is equal to 1 and shows that the running time of this algorithm is $\Omega(f(x))$

(c)

By pulling out the sum from the nested loop and keeping a partial sum we can significantly increase performance

```
for i=1 to n
s=A[i]
for j=i+1 to n
s+=A[j]
B[i,j] = s
end
end
```

The new performance of the loop contains 2 operations in the center instead of n we can rewrite our expression as $n((n-1)2) = 2n^2 - 2n$. With our new bound on the algorithm represented as $O(n^2)$. It is trivial to see that this solution is correct as it performs the exact same amount of insertions it just does not sum the numbers every single time before an insert.

3. *Why do we analyze the average-case performance of a randomized algorithm and not its worst-case performance?*

We do this because a randomized algorithm treats all inputs randomly thus increasing the chance that the input is the average case. This means that the algorithm then has a high probability of finishing in average case time. So to accurately describe the performance we use the average-case time because there is a high probability the algorithm will finish is average time given any input and a low chance that it will finish in worst case given the same input.